I can't pass an extremely competitive test to become a surgeon. But you give me any operation on a heart. I can perhaps do much better than most people. I am like an artist. Don't expect me to compete in an exam. Give me the job and I will show you how good I am.

NoSQL DB

Venkatesh Vinayakarao
venkateshv@cmi.ac.in
http://vvtesh.co.in

Chennai Mathematical Institute

The cost of managing traditional databases is high. Mistakes made during routine maintenance are responsible for 80 percent of application downtime. **– Dev Ittycheria, MongoDB.**

A Relation as a Data Model

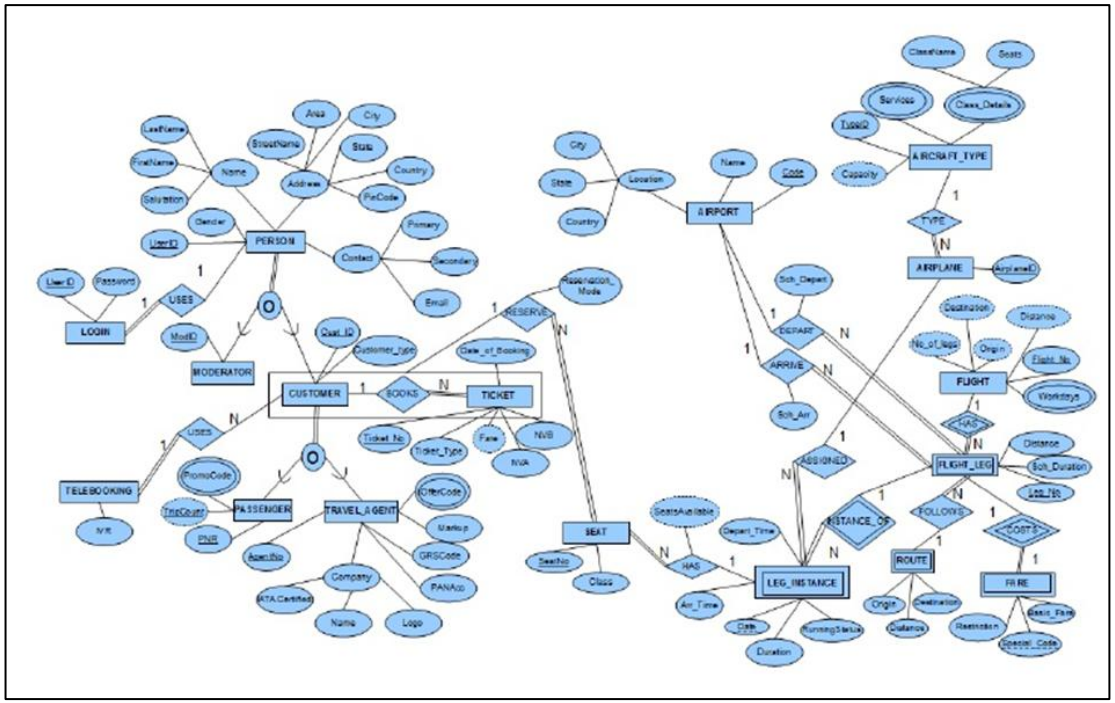
- Let the set, id = {1,2,3}
- Let the set, names = {vv, sd}
- What is id x names?
- We have a relation if we assign a sequential id to each name.

id	name
1	sd
2	vv

id	name
1	sd
1	vv
2	sd
2	vv
3	sd
3	vv

… and thus we had the relational database.

An Entity-Relationship Design



DB Designs:

- **Can get too complex!**
- **May become too hard to maintain!!**

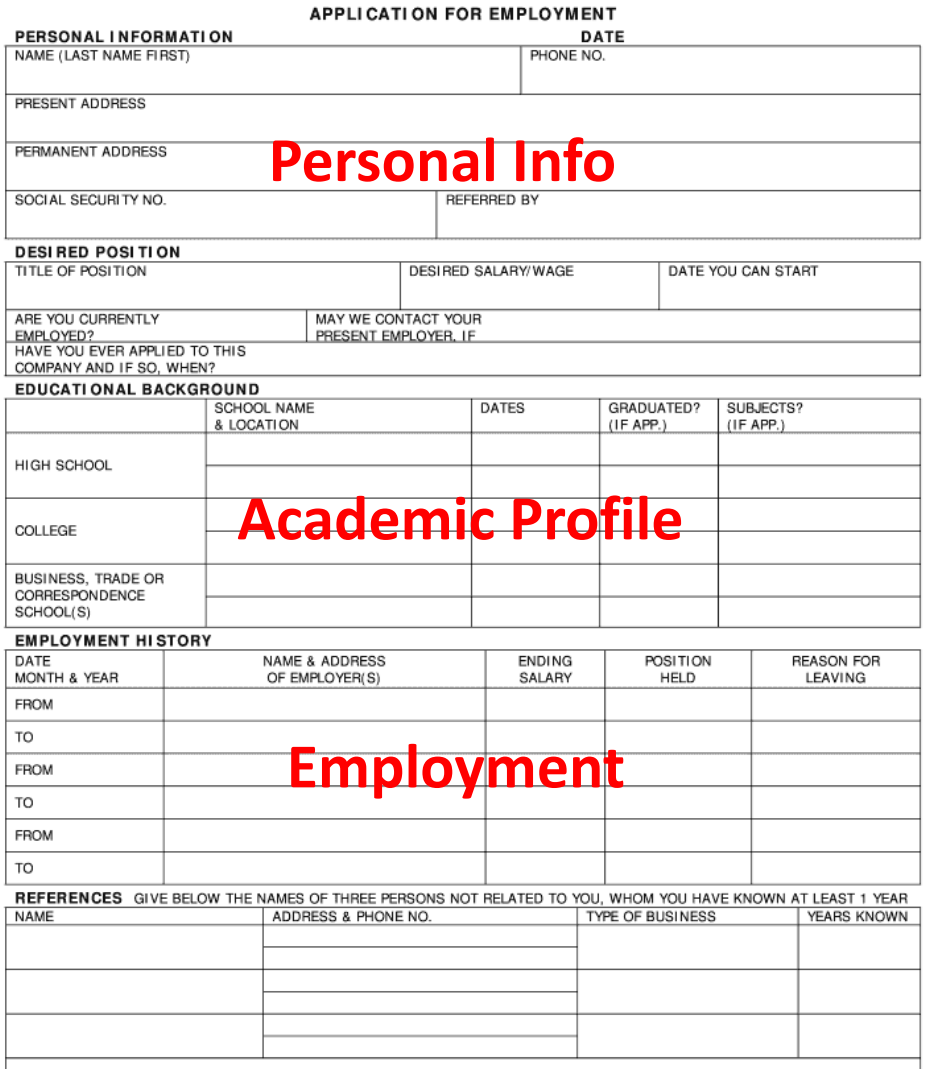
Source: https://www.slideshare.net/NarendranThangarajan/airline-reservation-system-db-design

Key Challenges of Relational DB

- Schema needs to be defined.
- Maintenance becomes harder over time.
- Impedance mismatch problem.
- Does not scale out by design.
- ACID Transactions – Consistency Vs. Availability Trade-off.

Impedance Mismatch

DB Design



APPLICATION FOR EMPLOYMENT

PERSONAL INFORMATION **DATE**

NAME (LAST NAME FIRST)	PHONE NO.
PRESENT ADDRESS	
PERMANENT ADDRESS	
SOCIAL SECURITY NO.	REFERRED BY

Personal Info

DESIRED POSITION

TITLE OF POSITION	DESIRED SALARY/WAGE	DATE YOU CAN START
ARE YOU CURRENTLY EMPLOYED?	MAY WE CONTACT YOUR PRESENT EMPLOYER, IF	
HAVE YOU EVER APPLIED TO THIS COMPANY AND IF SO, WHEN?		

EDUCATIONAL BACKGROUND

	SCHOOL NAME & LOCATION	DATES	GRADUATED? (IF APP.)	SUBJECTS? (IF APP.)
HIGH SCHOOL				
COLLEGE				
BUSINESS, TRADE OR CORRESPONDENCE SCHOOL(S)				

Academic Profile

EMPLOYMENT HISTORY

DATE MONTH & YEAR	NAME & ADDRESS OF EMPLOYER(S)	ENDING SALARY	POSITION HELD	REASON FOR LEAVING
FROM				
TO				
FROM				
TO				
FROM				
TO				

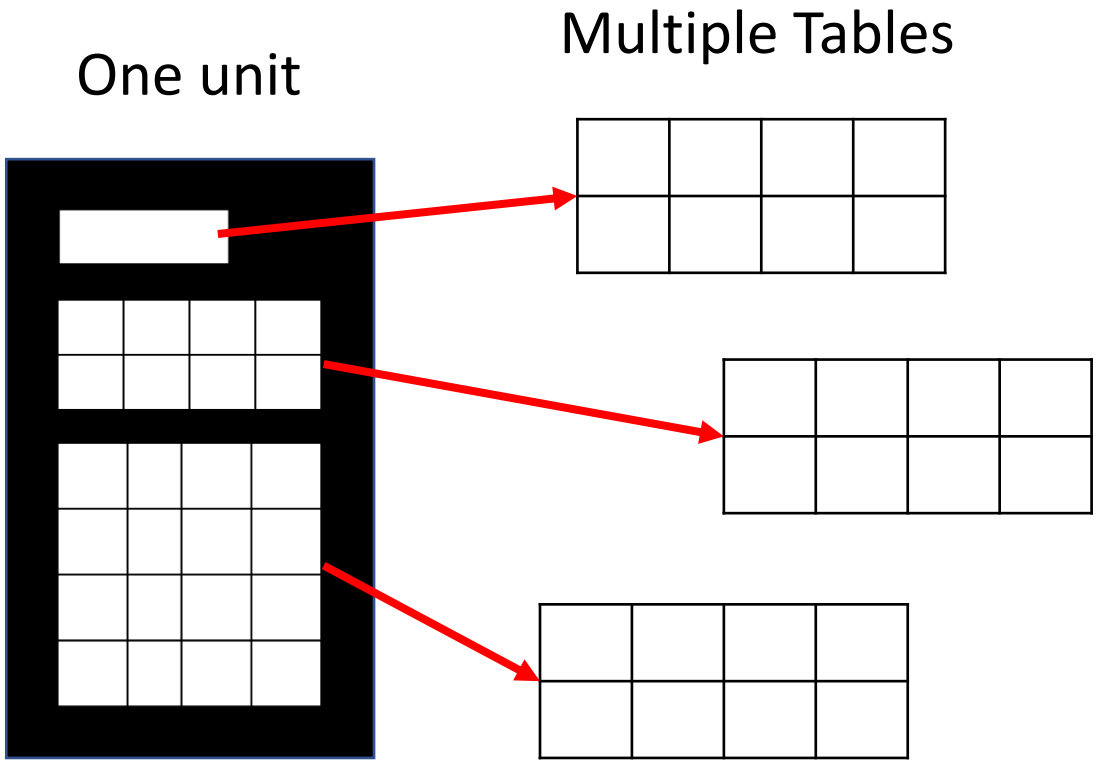
Employment

REFERENCES GIVE BELOW THE NAMES OF THREE PERSONS NOT RELATED TO YOU, WHOM YOU HAVE KNOWN AT LEAST 1 YEAR

NAME	ADDRESS & PHONE NO.	TYPE OF BUSINESS	YEARS KNOWN

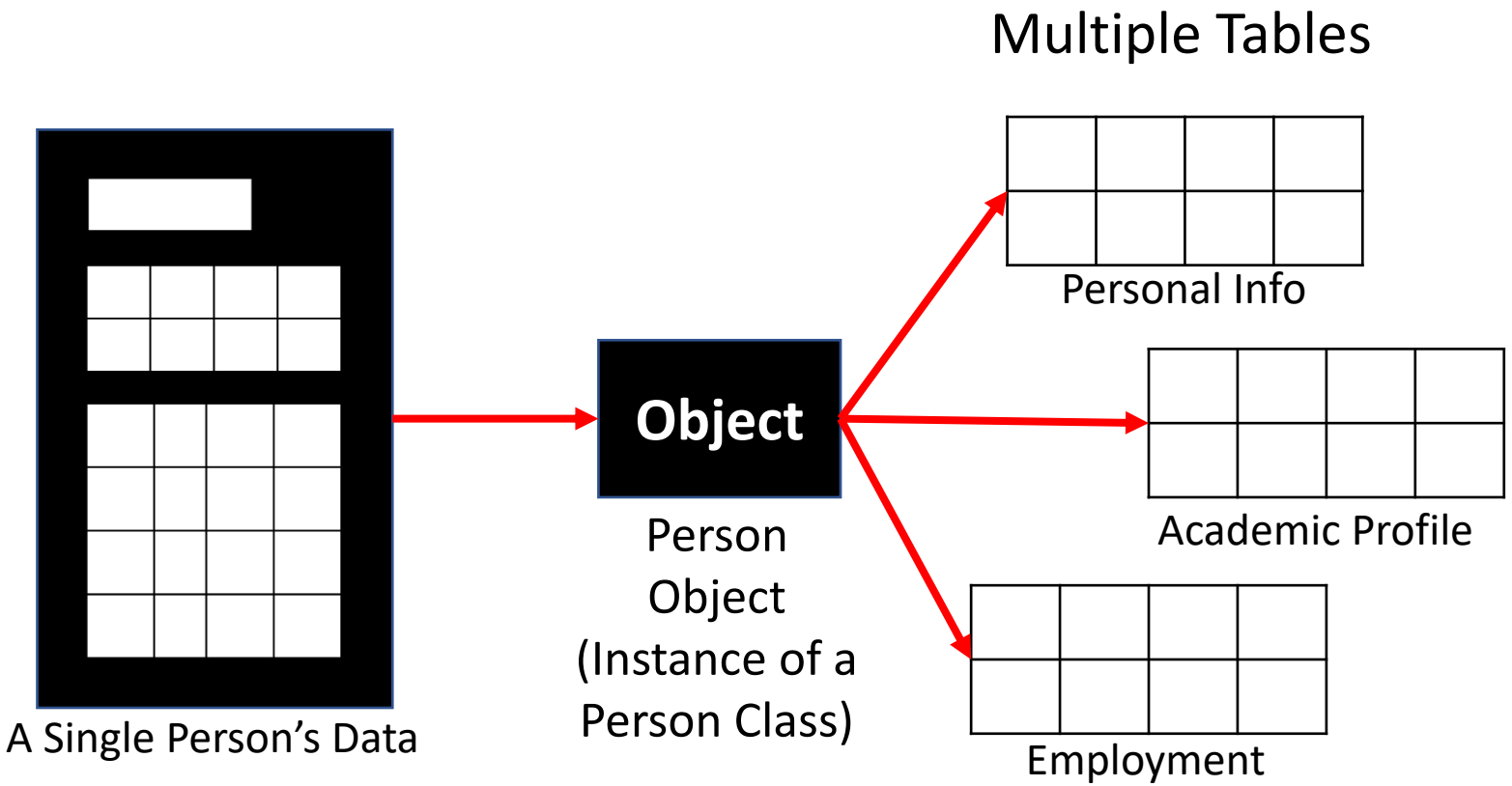
How will you design the DB for this content?

Impedance Mismatch Problem



Intermediate Solution: Object Relational Mapping (ORM)

Object Relational Mapping



Hibernate Framework, Java Data Objects, … and many other ORM frameworks emerged.

Scaling Out

Table Joins Using MapReduce

- How would you do it?

Map-side Join

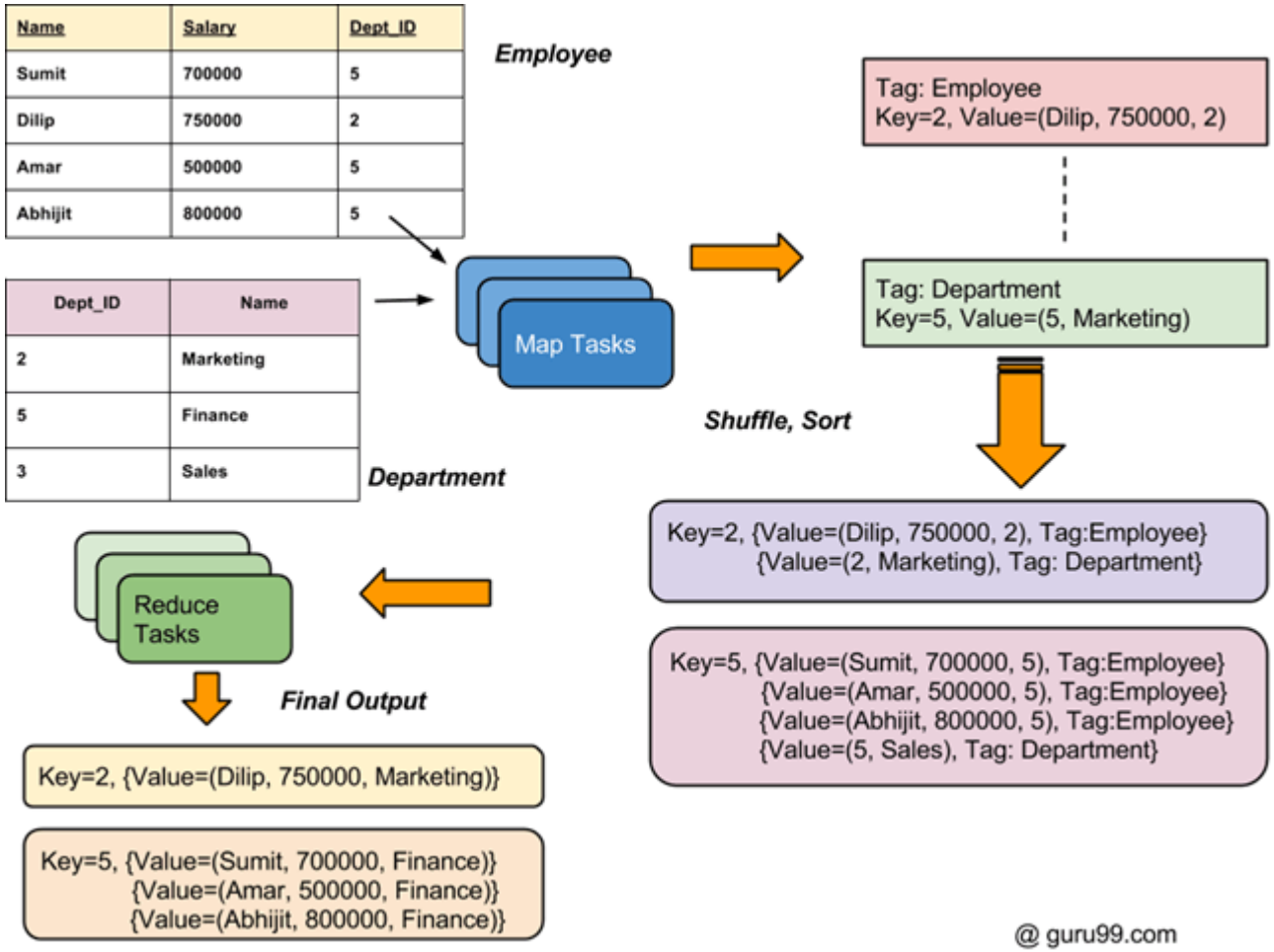
Join is performed by the mapper.

Reduce-side Join

Join is performed by the reducer.

Table joins are expensive. So, new solutions emerged. Google BigTable, Amazon Dynamo…

Join Pattern



See https://www.guru99.com/introduction-to-counters-joins-in-map-reduce.html

A New Movement was Born

- We needed a
  - Not only relational
  - Cluster friendly
  - Schemaless

  way to store and retrieve data.
- Johan Oskarsson proposed a meetup. He needed a twitter hashtag. He used, “**nosql**”.

NoSQL DB Types

Types of NoSQL DB

- Key-Value Stores
  - Simplest. Every item is a key-value pair.
  - Examples: Riak, Voldemort, and Redis
- Document DB
  - Complex data structures are represented as documents.
  - Examples: MongoDB
- Wide-Column Stores
  - Data stored as columns.
  - Examples: Cassandra and Hbase
- Graph DB
  - Examples: Neo4J and HyperGraphDB

Redis DB – Key Value Store

```
redis> GET nonexisting
(nil)
redis> SET mykey "Hello"
"OK"
redis> GET mykey
"Hello"
redis>
```

Read https://redis.io/commands/get

Voldemort DB

```
> bin/voldemort-shell.sh test tcp://localhost:6666
Established connection to test via tcp://localhost:6666
> put "hello" "world"
> get "hello"
version(0:1): "world"
> delete "hello"
> get "hello"
null
> help
...
> exit
k k thx bye.
```

mongoDB – Document Database

- mongoDB = “Hu**mongo**us DB”
  - Open-source
  - Document-based data model
  - Automatic scaling
  - Compromises on Availability (by default)

MongoDB vs. RDBMS

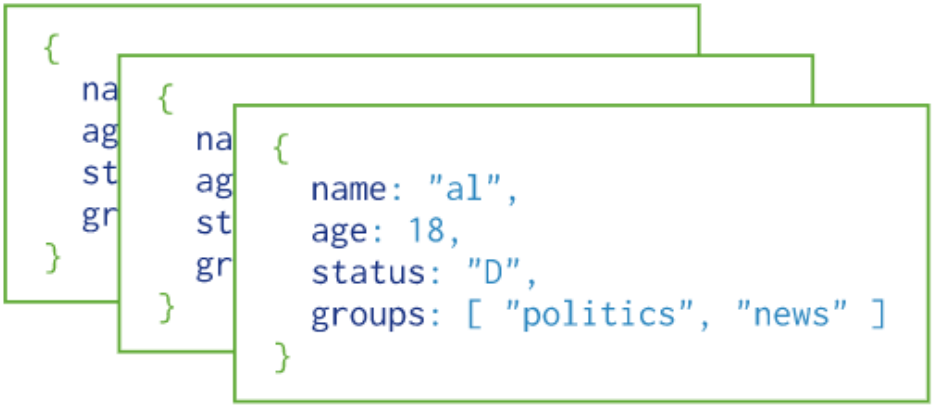
- Collection vs. table
- Document vs. row
- Field vs. column
- Schema-less

Document Data Model

- Documents are a natural way to represent data.
- Here is a “Person” object represented as a JSON document.
- MongoDB stores this as a BSON document (Binary representation of JSON).

```
{
  name: "sue",                          <-- field: value
  age: 26,                              <-- field: value
  status: "A",                          <-- field: value
  groups: [ "news", "sports" ]          <-- field: value
}
```

A record in MongoDB is a document



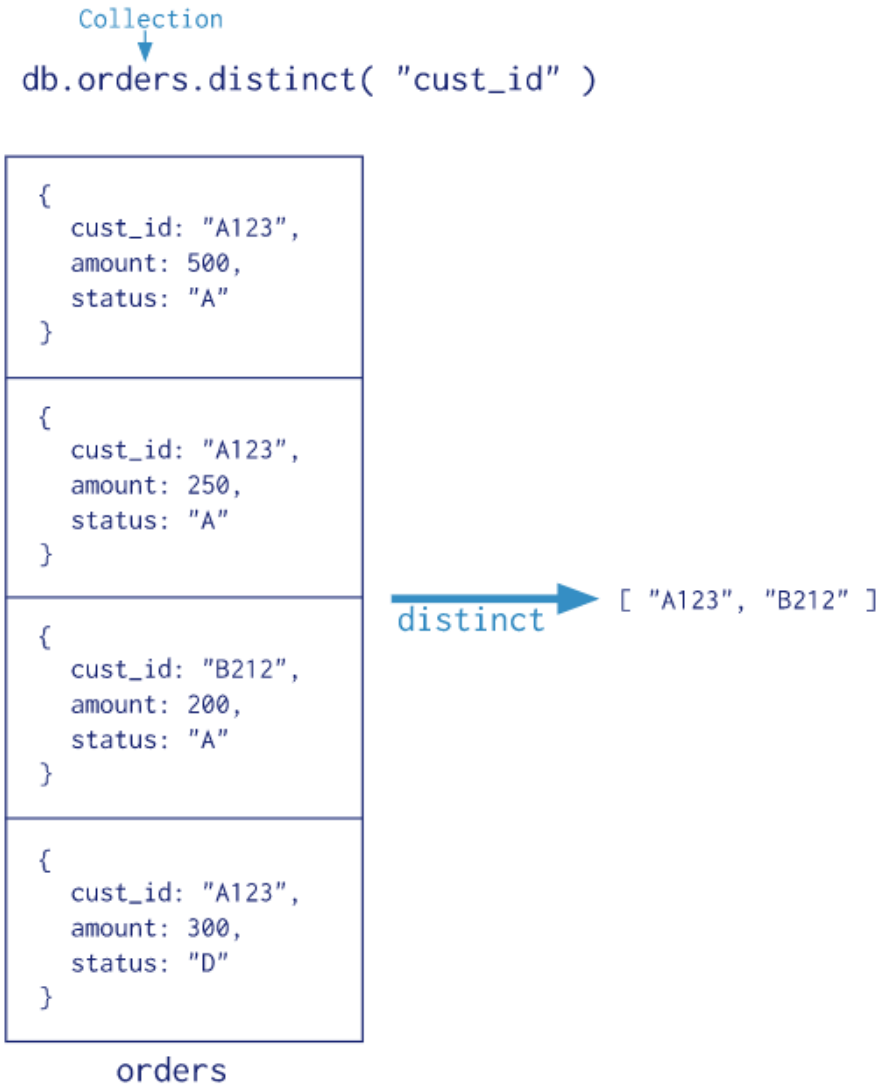

```
db.myNewCollection2.insertOne( { x: 1 } )
db.orders.deleteOne({ “name" : “al" } );
```

Commands

Read https://docs.mongodb.com/manual/core/databases-and-collections/

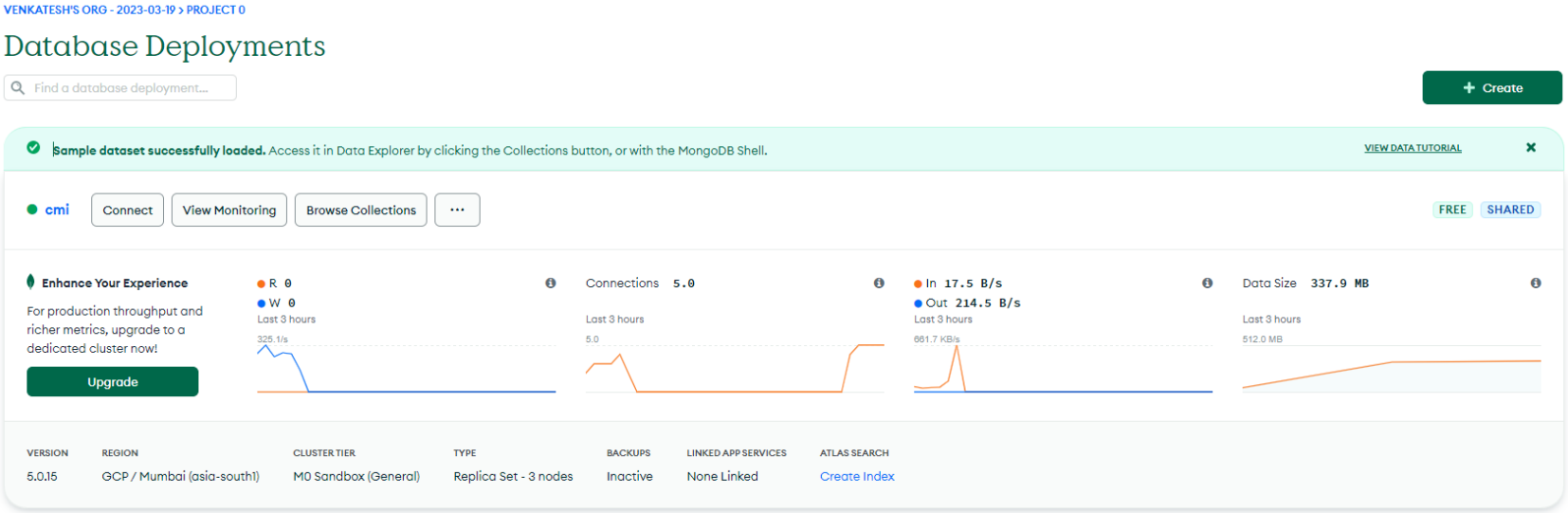
```
db.createCollection("personal_information")
db.personal_information.insertOne({
    name: "Venkatesh",
    address: "Church Street, Bangalore",
    email: "vv@cmi.comm",
    phone: "12345566"
})

db.personal_information.deleteOne({"name":"Venkatesh
"});
```

Operations on MongoDB Data



MongoDB Cloud Tutorial

- Visit cloud.mongodb.com, signup and login.
- Create a new database deployment.
- Name it “cmi”. Connect to it.



Connect to cmi

✔ Setup connection security > ✔ Choose a connection method > Connect

I do not have the MongoDB Shell installed | I have the MongoDB Shell installed

1 **Select your operating system and download the** mongosh

Windows

Download mongosh (1.8.0) or Copy download URL

2 **Add** `<your mongosh's download directory>/bin` **to your** `$PATH` **variable.** How to

3 **Run your connection string in your command line**

Use this connection string **in your application:**

```
mongosh "mongodb+srv://cmi.dhsa6tk.mongodb.net/myFirstDatabase" --apiVersion 1 --
username vvtesh
```

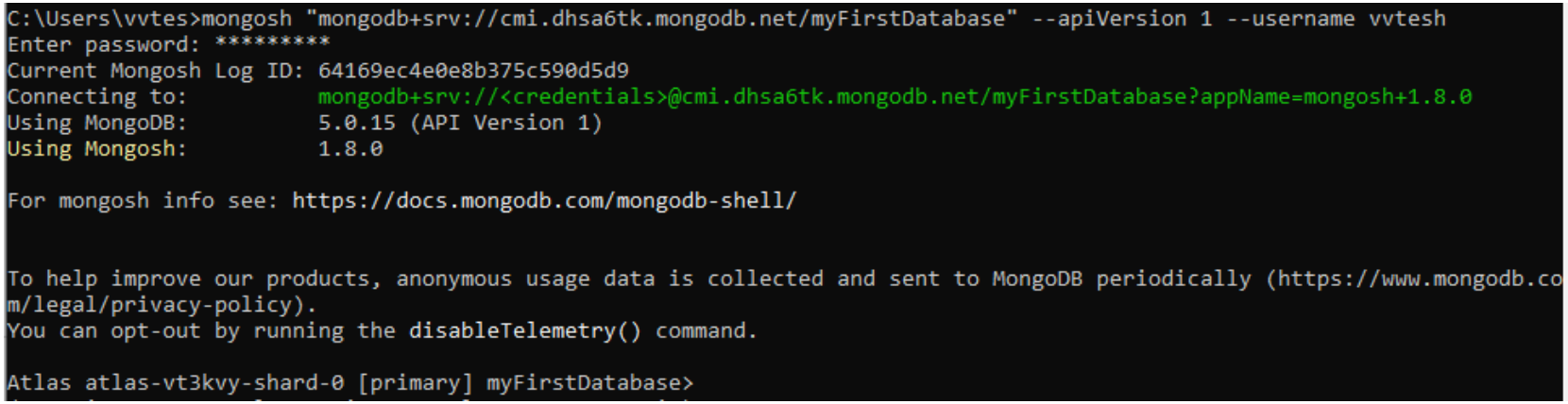
Replace **myFirstDatabase** with the name of the database that connections will use by default. You will be prompted for the password for the Database User, **vvtesh**. When entering your password, make sure all special characters are URL encoded.

Having trouble connecting? View our troubleshooting documentation

Go Back | Close

Open MongoDB Shell

- Follow instructions to open the mongodb shell.



Connection Error

- If you see this error, your mongodb cluster is down. Restart it from the cloud Database Deployments.

```
C:\Users\vvtes>mongosh "mongodb+srv://cmi.dhsa6tk.mongodb.net/" --apiVersion 1 --username cmi
Enter password: ***
Current Mongosh Log ID: 66004066c0d4dd9ff07bd929
Connecting to:          mongodb+srv://<credentials>@cmi.dhsa6tk.mongodb.net/?appName=mongosh+2.0.0
Error: querySrv ENOTFOUND _mongodb._tcp.cmi.dhsa6tk.mongodb.net
```

Try MongoDB Commands

- Use a DB and Create a Collection
  - show dbs
  - use <dbname>
  - show collections
  - db.createCollection(“students”)
- Manipulate the Collection
  - db.students.insert({ fname:“VV”, lname:“Rao” })
  - db.students.find()
  - db.students.find().count()
  - db.students.remove({fname:“VV”})
- Drop the Collection
  - db.students.drop()
- exit

See https://www.mongodb.com/basics/create-database

Insertion Error

- Note that “ and `` are not same.

```
Atlas atlas-vt3kvy-shard-0 [primary] myFirstDatabase> db.students.find()

Atlas atlas-vt3kvy-shard-0 [primary] myFirstDatabase> db.students.insert({ fname:“VV”, lname:“Rao” })
Uncaught:
SyntaxError: Unexpected character '“'. (1:27)

> 1 | db.students.insert({ fname:“VV”, lname:“Rao” })
    |                           ^
  2 |

Atlas atlas-vt3kvy-shard-0 [primary] myFirstDatabase> db.students.insert({ fname:"VV", lname:"Rao" })
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("6600435b7ee43387d352d8cd") }
}
Atlas atlas-vt3kvy-shard-0 [primary] myFirstDatabase>
```

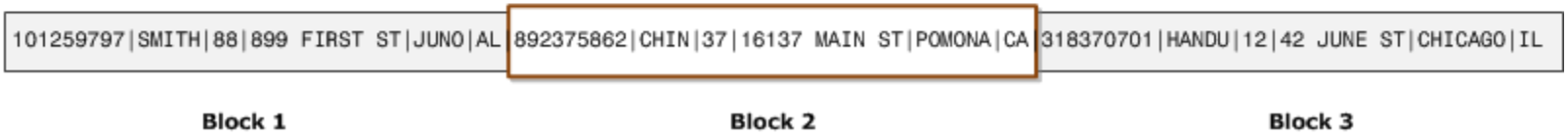
Quiz

How will you manage the data from a form that captures personal information (such as Name, address) in MongoDB?

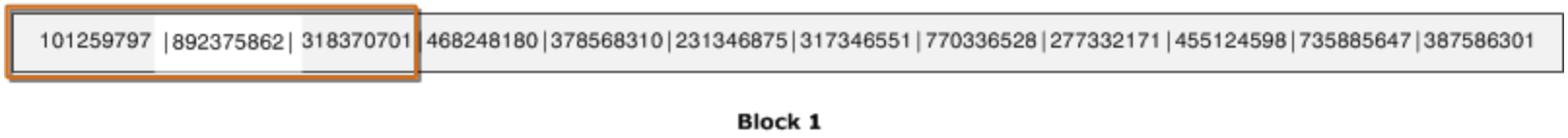
Write insert and delete queries. Describe the document structure.

Columnar Storage

SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL



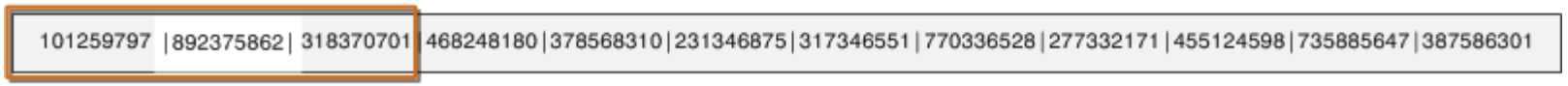
SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL



Read https://docs.aws.amazon.com/redshift/latest/dg/c_columnar_storage_disk_mem_mgmnt.html

Columnar Storage

SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL



Block 1

Same datatype in a block helps in devising efficient compression schemes. Therefore, improve storage efficiency.

Assumption: "***OLTP transactions*** *typically involve most or all of the columns in a row for a small number of records.* ***Data warehouse*** *queries commonly read only a few columns for a very large number of rows."*

Cassandra - Wide-Column Store

- A **column** is the basic data structure of Cassandra.
- A Column has three values, namely key or column name, value, and a time stamp.

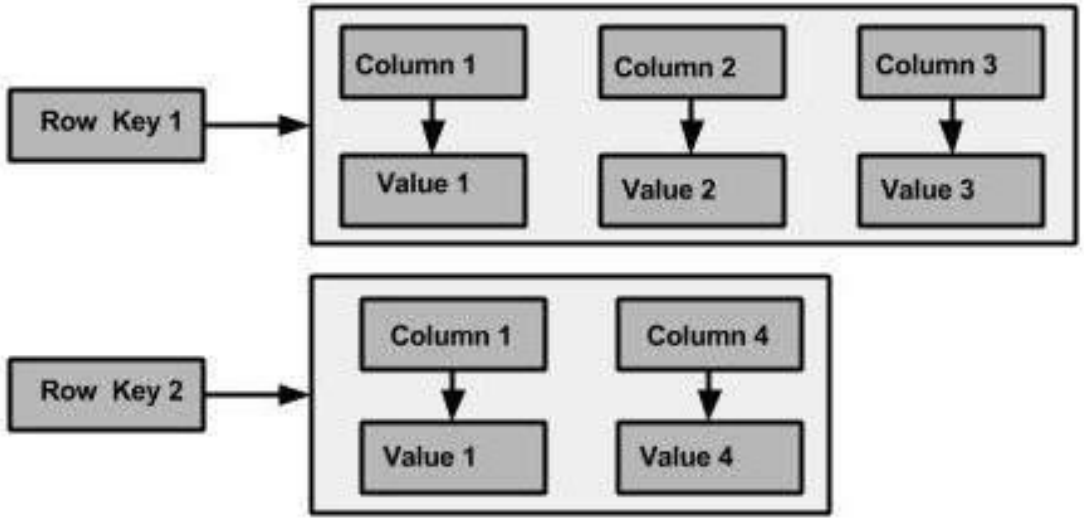
Column		
name : byte[]	value : byte[]	clock : clock[]

- A **super column** is a special column. stores a map of sub-columns.

Super Column	
name : byte[]	cols : map<byte[], column>

Column-Family DB

- Cassandra does not force individual rows to have all the columns.
- An example of a Cassandra column family:



Apache HBase Tutorial

- Start Cloudera
  - Start docker desktop
  - docker run --hostname=quickstart.cloudera --privileged=true -t -i --publish-all=true -p 8888:8888 -p 8080:80 -p 50070:50070 -p 8088:8088 -p 50075:50075 -p 8032:8032 -p 8042:8042 -p 9888:19888 cloudera/quickstart /usr/bin/docker-quickstart
- Start HBase Shell
  - hbase shell
- Note
  - Each value is stored as rowkey + columnfamily + columnqualifier + datetime + value.

Apache HBase Tutorial

- Try the following commands on cloudera quickstart console
  - Creates Student table with three column families.
    - create 'Student', 'personal_data','academic_data','other_data’
  - List all the tables
    - list
  - Insert a student with rowkey S101 and “John” as the name in personal_data
    - put 'Student','S101','personal_data:name','John'
      - Syntax is put ’<table name>’,’rowkey’,’<colfamily:colname>’,’<value>’
  - Insert more data
    - put 'Student','S101','personal_data:address','#145, NewRoad,Chennai'
    - put 'Student','S101','academic_data:class','Course A'
    - put 'Student','S101','academic_data:year','second'

Apache HBase Tutorial

- See the entered data
  - get 'Student','S101'
  - scan 'Student'
- How many rows do you have?
  - count 'Student'
- Clean up
  - delete 'Student','S101','academic_data:class'
  - disable 'Student'
  - drop 'Student'

Apache HBase Tutorial

```
hbase(main):001:0> create 'Student' , 'personal_data','academic_data','other_data'
0 row(s) in 2.6160 seconds

=> Hbase::Table - Student
hbase(main):002:0> list
TABLE
Student
1 row(s) in 0.0230 seconds
```

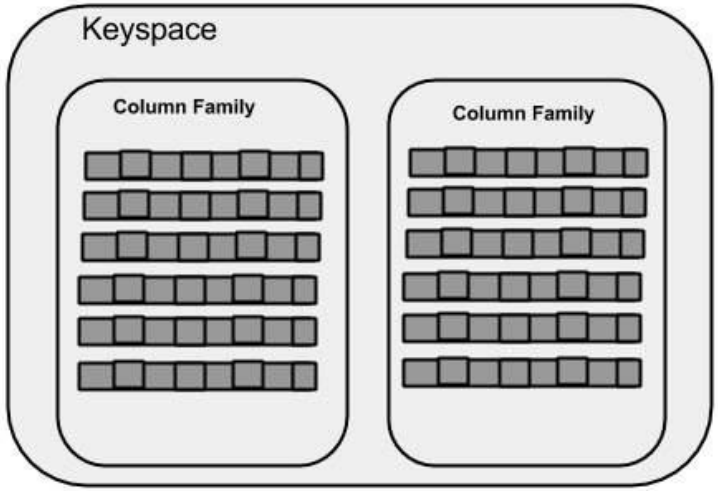
```
hbase(main):003:0> put 'Student','S101','personal_data:name','John'
0 row(s) in 0.1710 seconds

hbase(main):004:0> put 'Student','S101','personal_data:address','#145, NewRoad,Chenna
i'
0 row(s) in 0.0180 seconds

hbase(main):005:0> put 'Student','S101','academic_data:class','Course A'
0 row(s) in 0.0110 seconds

hbase(main):006:0> put 'Student','S101','academic_data:year','second'
0 row(s) in 0.0180 seconds
```

```
hbase(main):015:0> disable 'Student'
0 row(s) in 2.4290 seconds

hbase(main):016:0> drop 'Student'
0 row(s) in 2.3430 seconds

hbase(main):017:0> list
TABLE
0 row(s) in 0.0340 seconds
```

```
hbase(main):010:0> delete 'Student','S101','academic_data:class'
0 row(s) in 0.0930 seconds

hbase(main):011:0> scan 'Student'
ROW                     COLUMN+CELL
 S101                   column=academic_data:year, timestamp=1697422901904, value=seco
                        nd
 S101                   column=personal_data:address, timestamp=1697422885588, value=#
                        145, NewRoad,Chennai
 S101                   column=personal_data:name, timestamp=1697422867789, value=John
1 row(s) in 0.0510 seconds
```

Cassandra Keyspace

- Keyspace is a container for a list of one or more column families.
- A column family, in turn, is a container of a collection of rows.
- Each row contains ordered columns.



cqlsh

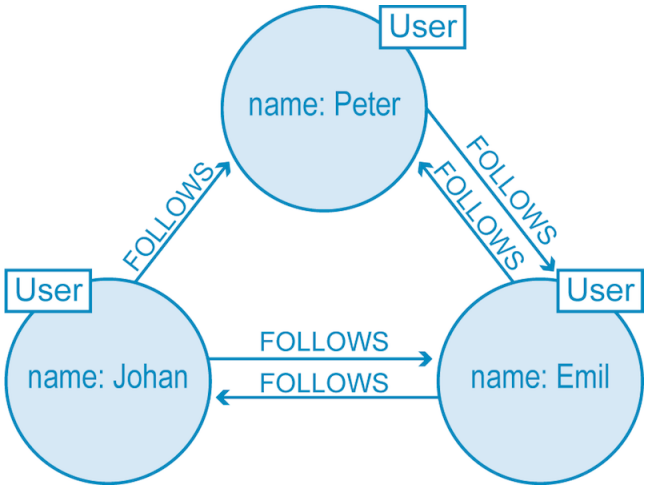
- Cassandra Query Language Shell

```
[hadoop@linux bin]$ cqlsh
Connected to … Cluster at ….
cqlsh>  select * from emp;
```

- Note: Cassandra does not join!
- If you need to lookup several tables, create another column-family.

Graph DB

- Facebook, LinkedIn, Google …have connected data.
- It is natural to store and retrieve data as graphs.



Twitter users represented in a graph database model.

Read https://neo4j.com/blog/why-graph-databases-are-the-future/

Transactions, Consistency and CAP Theorem

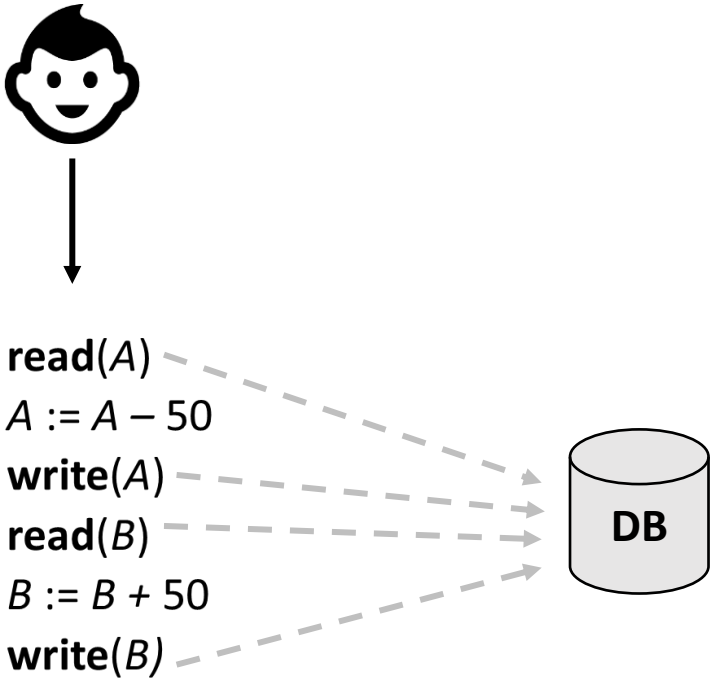
Transaction

1. **read**(*A*)
2. *A* := *A* – 50
3. **write**(*A*)
4. **read**(*B*)
5. *B* := *B* + 50
6. **write**(*B*)

transfer $50 from account A to account B

A transaction is a *unit* of program execution that accesses and possibly updates various data items.

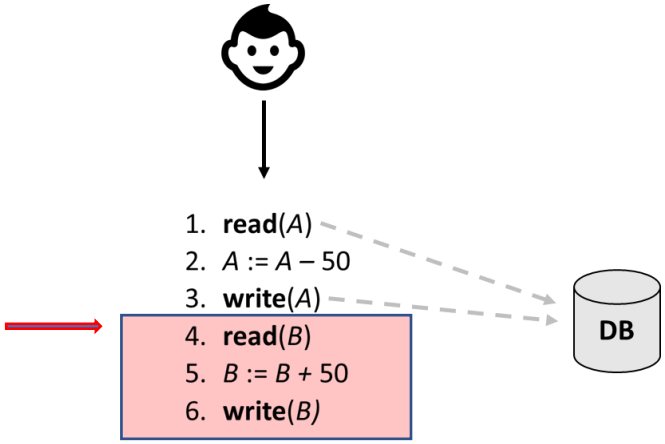
Do You See Any Issues Here?



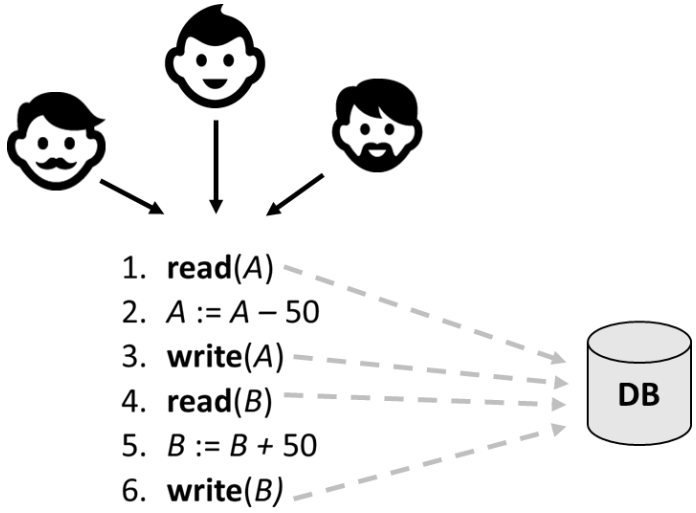
A transaction that reads
and writes to disk.

Issues

- Two main issues to deal with:



Failure (hardware failure, system crash, software defect…)



concurrent execution

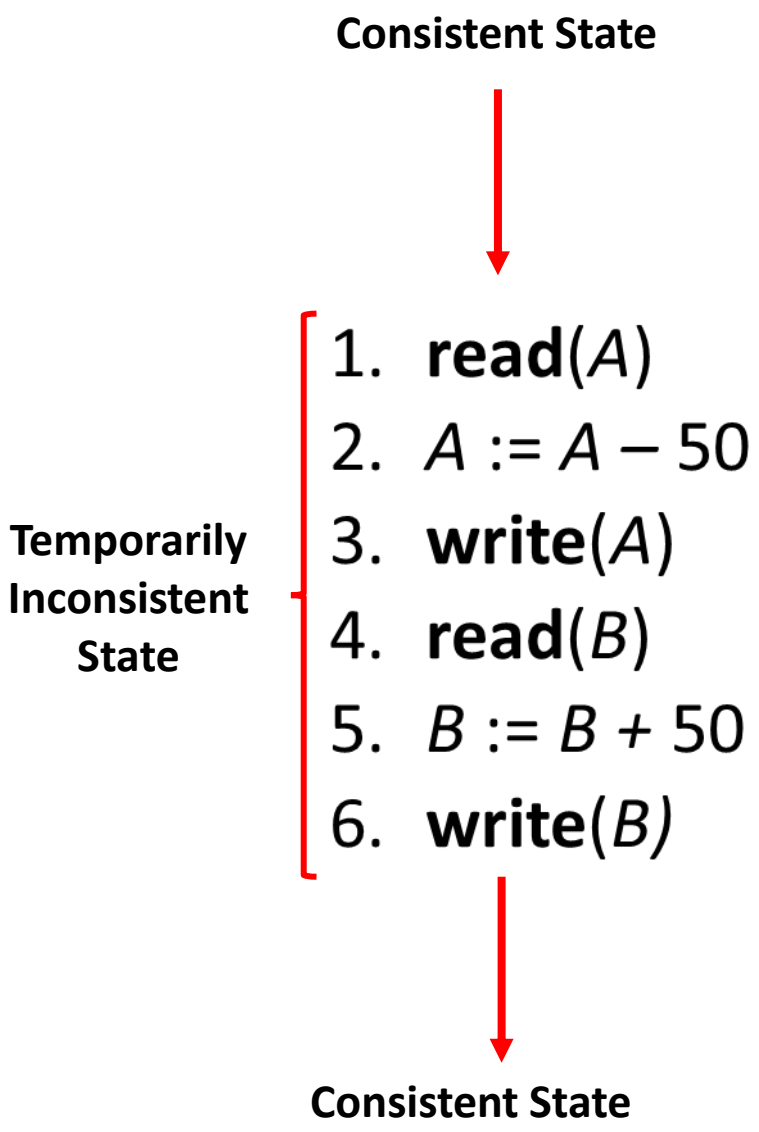
Atomicity

- **What happens if step 3 is executed but not step 6?**
  - Failure could be due to software or hardware
- The system should ensure that updates of a partially executed transaction are not reflected in the database.

1. **read**(A)
2. $A := A - 50$
3. **write**(A)
4. **read**(B)
5. $B := B + 50$
6. **write**(B)

Consistency

- Respect
  - Explicitly specified integrity constraints
  - Implicit integrity constraints
    - e.g., sum of balances of all accounts stays constant



Isolation

- T2 sees an inconsistent database if T1 and T2 are concurrent.

T1	T2
1. **read**(*A*)	
2. *A* := *A* – 50	
3. **write**(*A*)	
	read(A), read(B), print(A+B)
4. **read**(*B*)	
5. *B* := *B* + 50	
6. **write**(*B*)	

- Isolation can be ensured trivially by running transactions **serially**
  - That is, one after the other.

Durability

- After step 6, the updates to the database by the transaction must
  - persist even if there are software or hardware failures.

1. **read**(A)
2. $A := A - 50$
3. **write**(A)
4. **read**(B)
5. $B := B + 50$
6. **write**(B)

ACID Properties

- **Atomicity.** Either all operations of the transaction are properly reflected in the database or none are.
- **Consistency.** A transaction must bring the database from one valid state to another, ensuring data integrity and adhering to predefined rules and constraints
- **Isolation.** Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.
  - That is, for every pair of transactions T_i and T_j, it appears to T_i that either T_j finished execution before T_i started, or T_j started execution after T_i finished.
- **Durability.** After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

How do we manage ACID properties in the world of distributed databases?

"A ***distributed database*** is a collection of multiple, logically interrelated databases distributed over a computer network. A distributed DBMS is then defined as the software system that permits the management of the distributed database and makes the distribution transparent to the users." - Özsu and Valduriez, Principles of Distributed Database Systems.

But, as a facebook user, I had a different observation…

Eventual Consistency

- I updated my facebook status and asked my friend to check it out.
- But she found nothing there!!!
- Asked her to wait a bit and check again.
- Now, she finds it!

Eventual Consistency

- Facebook is eventually consistent.
- Why not use a strongly consistent model?
  - Stores Petabytes of data.
  - We have Availability vs. Consistency tradeoff.

Read https://www.cs.umd.edu/~abadi/papers/abadi-pacelc.pdf

CAP Theorem

- Concerns while designing distributed systems:
  - **Consistency** –all clients of a data store get responses to requests that ‘make sense’. For example, if Client A writes 1 and later 2 to location X, Client B cannot read 2 followed by 1.
  - **Availability** – all operations on a data store eventually return successfully. We say that a data store is ‘available’ for, e.g. write operations.
  - **Partition tolerance** – if the network stops delivering messages between two sets of servers, will the system continue to work correctly?

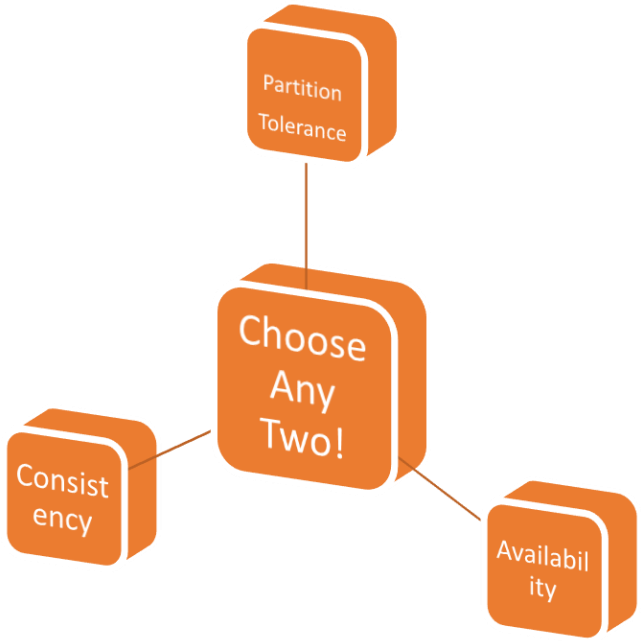
The CAP Message

If you:

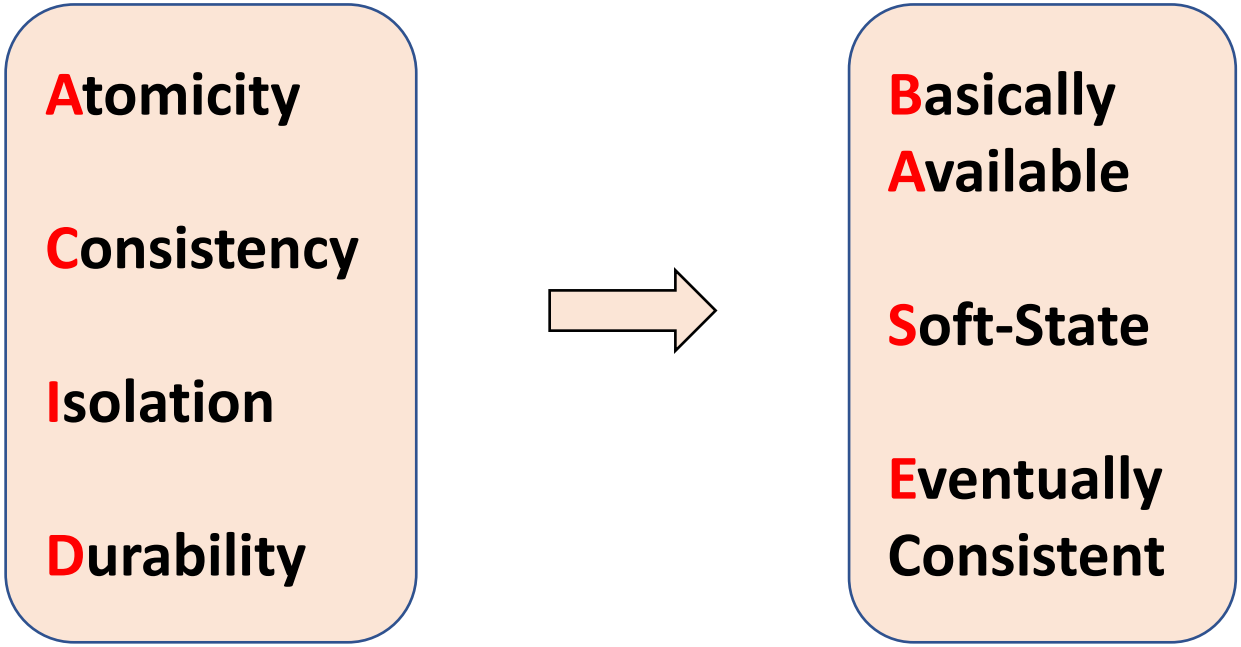
- cannot limit the number of faults,
- requests can be directed to any server, and
- insist on serving every request you receive,

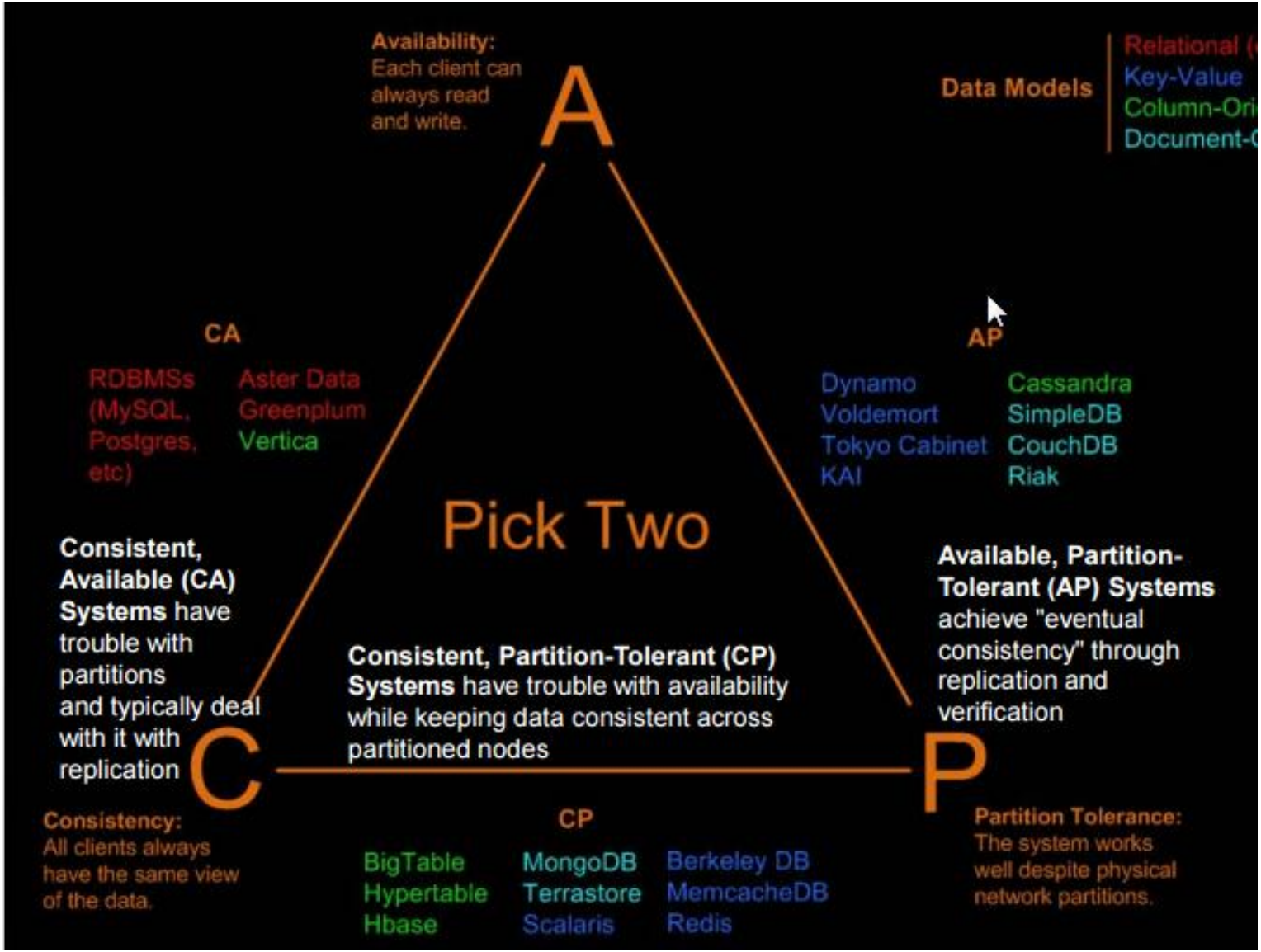
Then:

- you cannot possibly be consistent.



The Transaction Properties

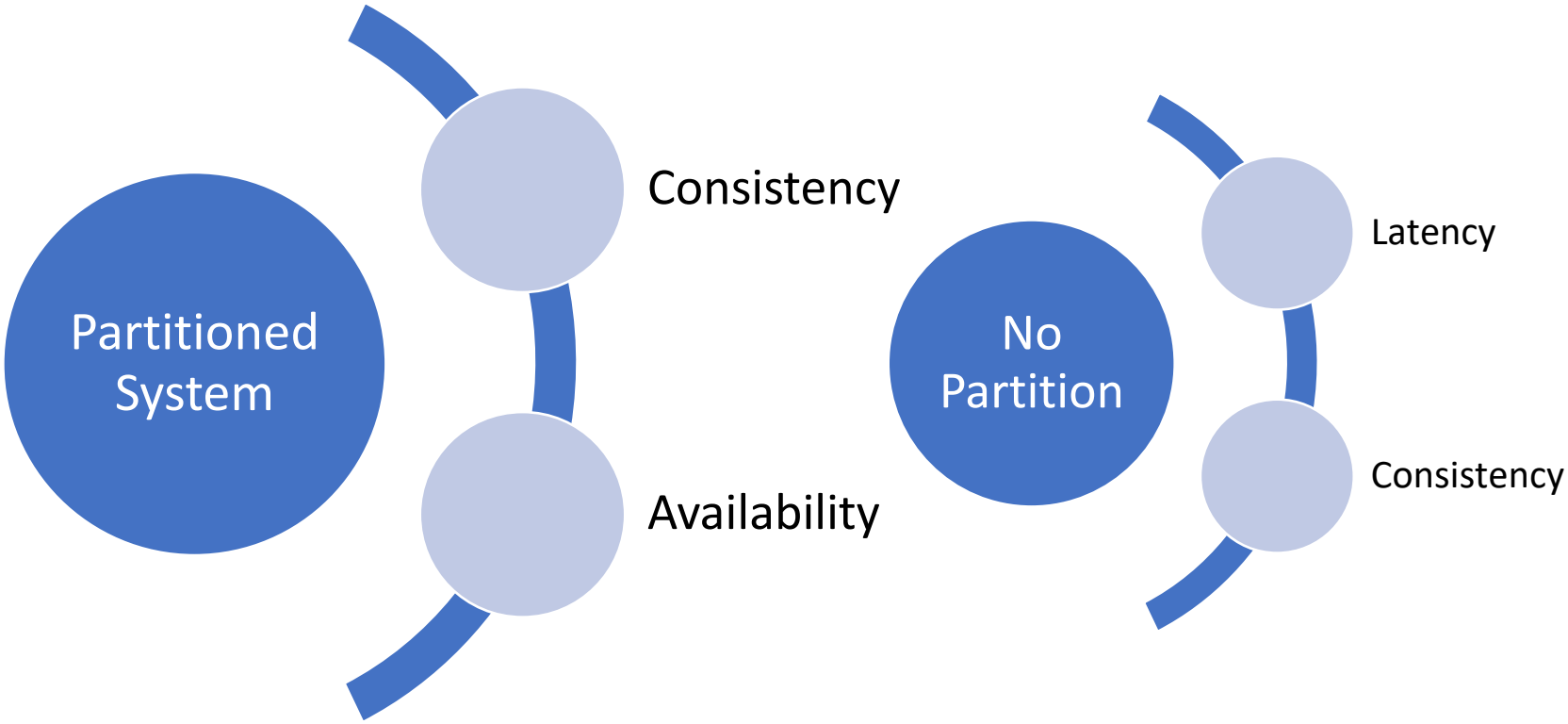



Availability:
Each client can always read and write.
A
Data Models
Relational (
Key-Value
Column-Ori
Document-
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Pick Two
Consistent, Available (CA) Systems have trouble with partitions and typically deal with it with replication
Consistent, Partition-Tolerant (CP) Systems have trouble with availability while keeping data consistent across partitioned nodes
Available, Partition-Tolerant (AP) Systems achieve "eventual consistency" through replication and verification
C
P
Consistency:
All clients always have the same view of the data.
CP
BigTable
Hypertable
Hbase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance:
The system works well despite physical network partitions.

PACELC (pronounced “pass-elk”)

- PACELC extends the CAP theorem
  - If system is partitioned (P),
    - Choose between Consistency (C) and Availability (A)
  - Else (E)
    - In the absence of partitions, choose between latency (L) and consistency (C).

PACELC



https://www.cs.umd.edu/~abadi/papers/abadi-pacelc.pdf

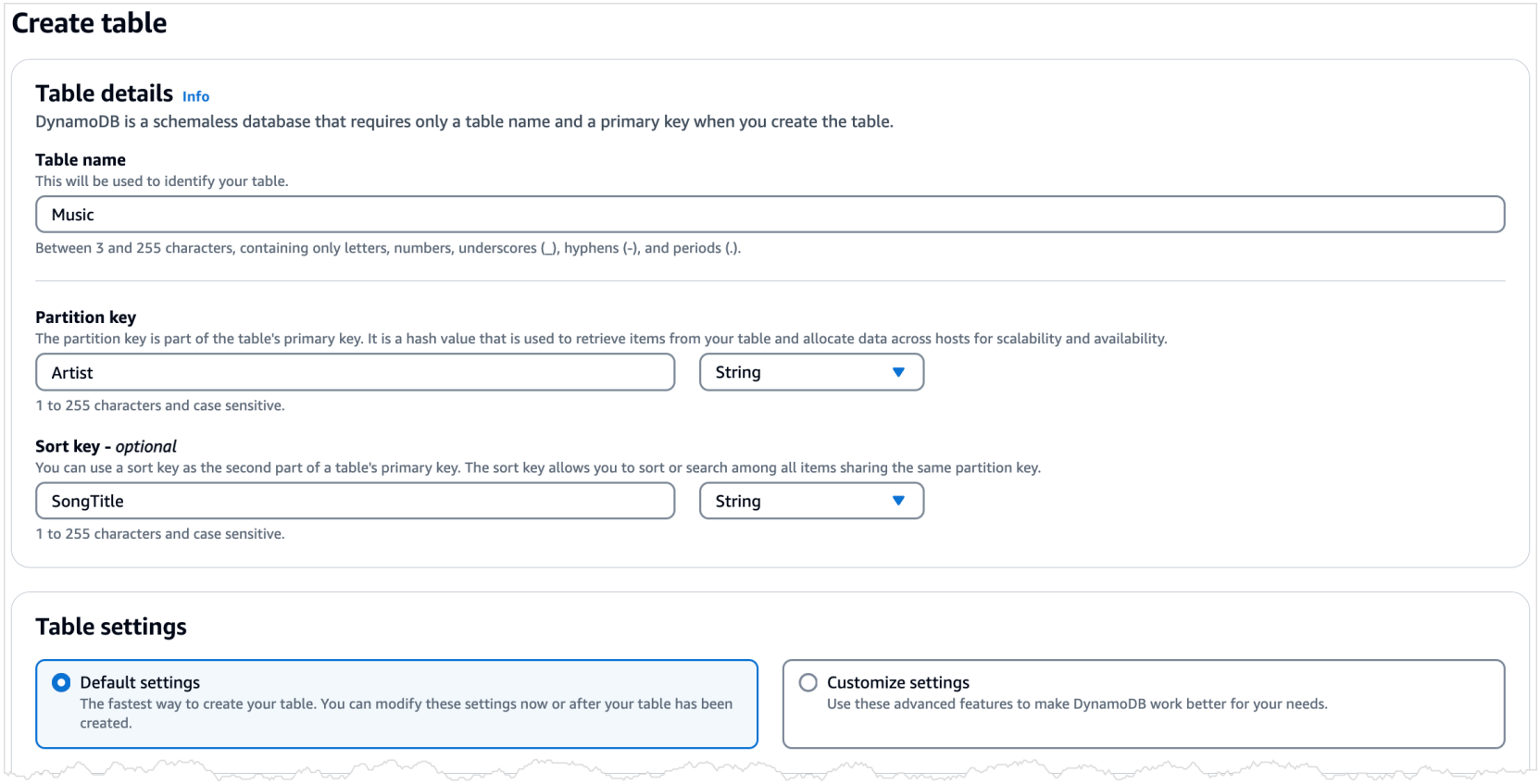
A prominent nosql db…

Amazon DynamoDB

- NoSQL
- Fully managed database
  - 99.999% availability SLA
  - No maintenance, No upgrades, No patching
- With single-digit millisecond performance at any scale
  - Scale to zero! Or Scale as much as you wish.

Watch https://www.youtube.com/watch?v=TCnmtSY2dFM

Fully Managed



Source: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/getting-started-step-1.html

Tables, items, and attributes

Table

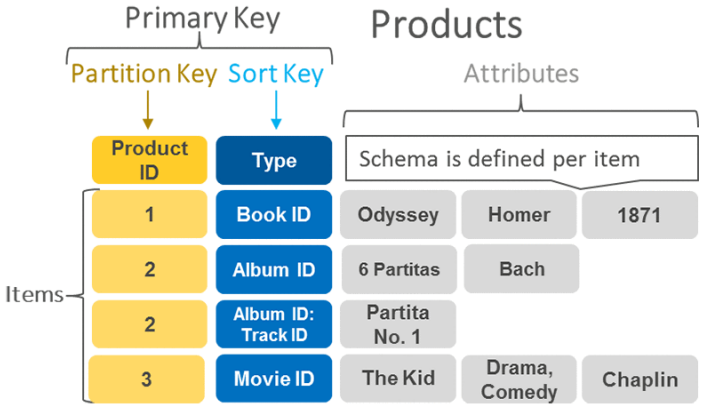
People

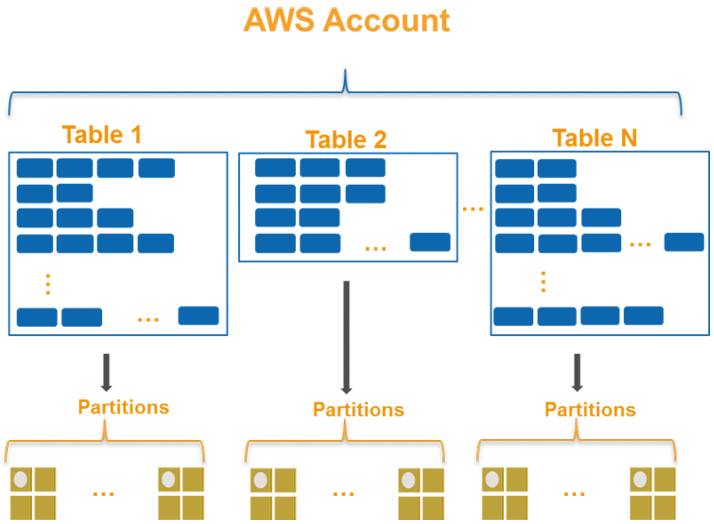
Primary key Partition key: PersonID	Attributes			
101	LastName	FirstName	Phone	
	Smith	Fred	555-4321	
102	LastName	FirstName	Address	
	Jones	Mary	{"Street":"123 Main","City":"Anytown","State":"OH","ZIPCode":12345}	
103	LastName	FirstName	FavoriteColor	Address
	Stephens	Howard	Blue	{"Street":"123 Main","City":"London","PostalCode":"ER3 5K8"}

Items

Keys

- Partition Key
  - Items are distributed across 10-GB storage units, called partitions (physical storage internal to DynamoDB)
- Sort Key
  - All data under a partition key is sorted by the sort key value.





Read/Write Data

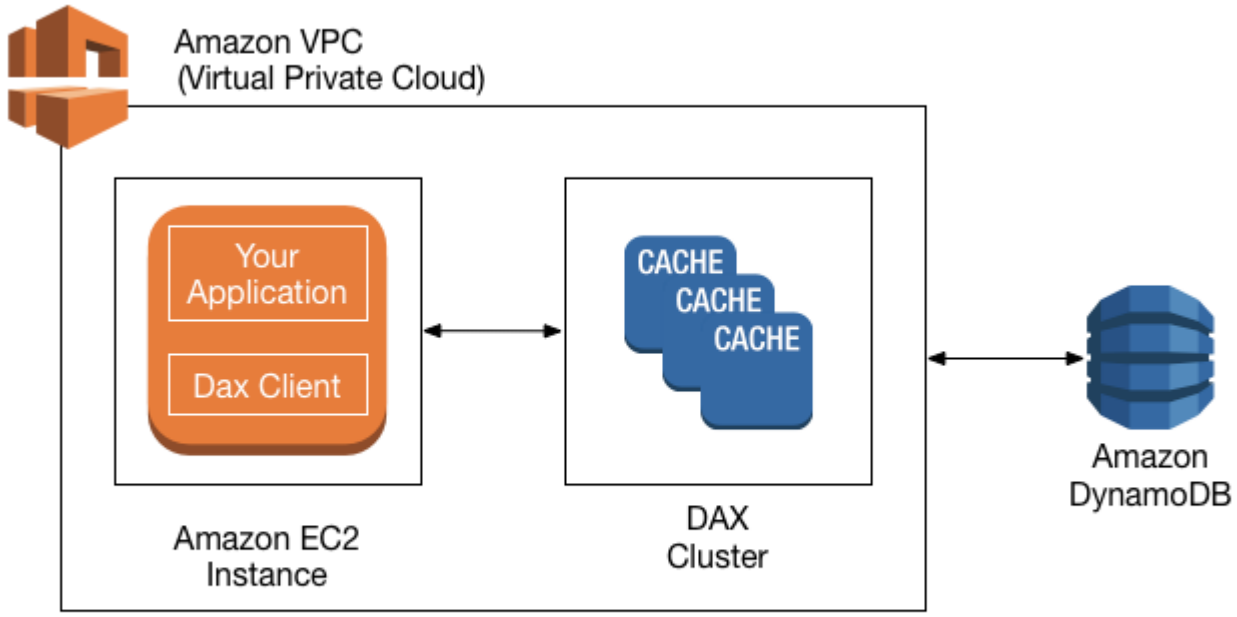
```
Region region = Region.US_EAST_1;

DynamoDbClient ddb = DynamoDbClient.builder()
                                   .region(region) .build();

putItemInTable(ddb, tableName, key, keyVal,
albumTitle, albumTitleValue, awards, awardVal,
songTitle, songTitleVal);

ddb.close();
```

Query Data

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression "Artist = :name" \
    --expression-attribute-values  '{":name":{"S":"Acme Band"}}'
```

In-memory Acceleration with DAX

- Amazon DynamoDB Accelerator (DAX)



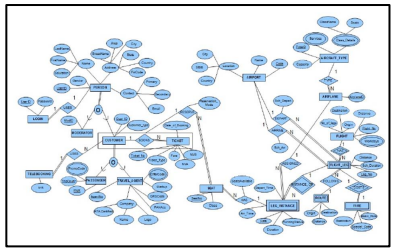
DynamoDB Streams

- Similar to “Triggers” in the RDBMS world
- Supports event-driven programming
  - With triggers, you can build applications that react to data modifications in DynamoDB tables.

DynamoDB: Key Features

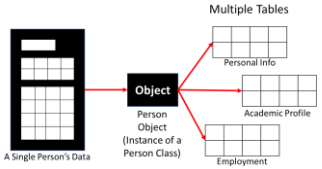
- Configurable to achieve either eventual consistency (by default) or strong consistency.
- Supports Transactions

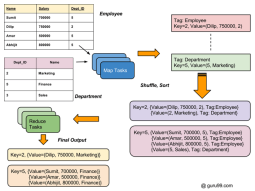
Summary



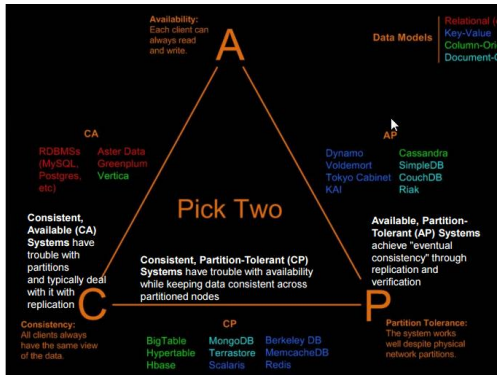
Schema-based Relational Model - maintenance problems

Impedance Mismatch





Scale-up Challenges



CAP Theorem

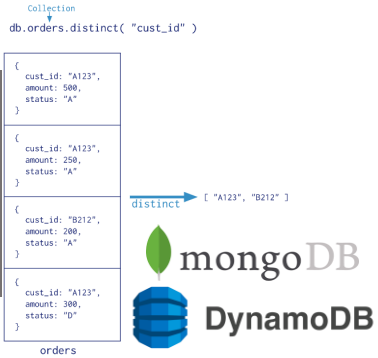
Types of NoSQL datastores

Key-Valuecv

```
redis> GET nonexisting
(nil)
redis> SET mykey "Hello"
"OK"
redis> GET mykey
"Hello"
redis>
```

redis

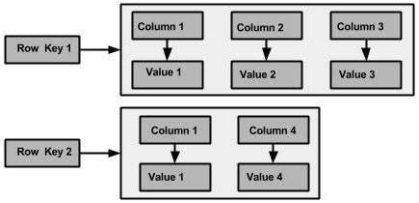
Doc-based



mongoDB

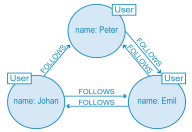
DynamoDB

Columnar DB



Apache CASSANDRA

Graph DB



neo4j

Thank You